NASA/CR–2017-219371

Formal Methods Tool Qualification

Lucas G. Wagner, Darren Cofer, and Konrad Slind
Rockwell Collins, Inc., Cedar Rapids, Iowa

Cesare Tinelli and Alain Mebsout
University of Iowa, Iowa City, Iowa

February 2017

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/CR–2017-219371

Formal Methods Tool Qualification

Lucas G. Wagner, Darren Cofer, and Konrad Slind
Rockwell Collins, Inc., Cedar Rapids, Iowa

Cesare Tinelli and Alain Mebsout
University of Iowa, Iowa City, Iowa

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NNL14AA06C

February 2017

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Table of Contents

1 Executive Summary

Formal methods tools have been shown to be effective at finding defects in safety-critical digital systems including avionics systems. The publication of DO-178C and the accompanying formal methods supplement DO-333 allows applicants to obtain certification credit for the use of formal methods without justification as an alternative method.

However, there are still many issues that must be addressed before formal verification tools can be injected into the design process for digital avionics systems. For example, DO-333 identifies three categories of formal methods tools: theorem provers, model checkers, and abstract interpretation tools. Most developers of avionics systems are unfamiliar with even the basic notions of formal verification, much less which tools are most appropriate for different problem domains. Different levels of expertise and skills will have to be mastered to use these tools effectively. DO-333 also requires applicants to provide evidence of soundness for any formal method to be used; i.e., that it will never provide unwarranted confidence. Constructing a soundness argument is not something most practicing engineers understand. Finally, DO-178C requires that a tool used to meet DO-178C objectives be qualified in accordance with the tool qualification document DO-330. While formal verification tools only need to meet the objectives for the lower qualification levels (TQL-4 or TQL-5), it is not unreasonable to expect their qualification to pose unique challenges.

In this project we have accomplished the following tasks related to qualification of formal methods tools:

1. Investigate the assurances that are necessary and appropriate to justify the application of formal methods tools throughout all phases of design in real safety-critical settings.
2. Produce practical examples of how to qualify formal verification tools in each of the three categories identified in DO-333.
3. Explore alternative approaches to the qualification of formal methods tools.

We have conducted an extensive study of existing formal methods tools, identifying obstacles to their qualification and proposing mitigations for those obstacles. We have produced a qualification plan for an open source formal verification tool, the Kind 2 model checker. We have investigated the feasibility of independently verifying tool outputs through the generation of proof certificates which are then verified by an independent certificate checking tool that verifies them. Finally, we have investigated the feasibility of qualifying this proof certificate checker in lieu of qualifying the model checker itself.

2 Introduction

Civilian aircraft must undergo a rigorous ***certification*** process to establish their *airworthiness*. Airworthiness is a measure of an aircraft's suitability to operate safely in its intended environment, the National Airspace System (NAS). Certification encompasses the entire aircraft and all of its components. This includes the engines, the airframe, the landing and taxi systems, and flight deck display systems, to name a few. Guidelines for developing certifiable aircraft are provided in *APR4754A: Guidelines for Development of Civil Aircraft and Systems* [1]. Similarly, guidance for addressing aspects of safety in civilian aircraft are provided in *ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment* [2]. Together these documents provide guidance for developing safe and trustworthy civilian aircraft that can fly in the NAS. However, they only address the aircraft and its components at the system level. They do not provide guidance on the hardware and software aspects of an avionics system. Instead, prospective applicants are instructed to consider additional documents detailing the aspects of software and hardware separately. For hardware, the appropriate document is *Design Assurance Guidance for Airborne Electronic Hardware* [3], also known as DO-254. For software, the document is *Software Considerations in Airborne Systems and Equipment Certification* [4], also known as DO-178C. The document tree for civilian avionics certifications is shown in Figure 1.

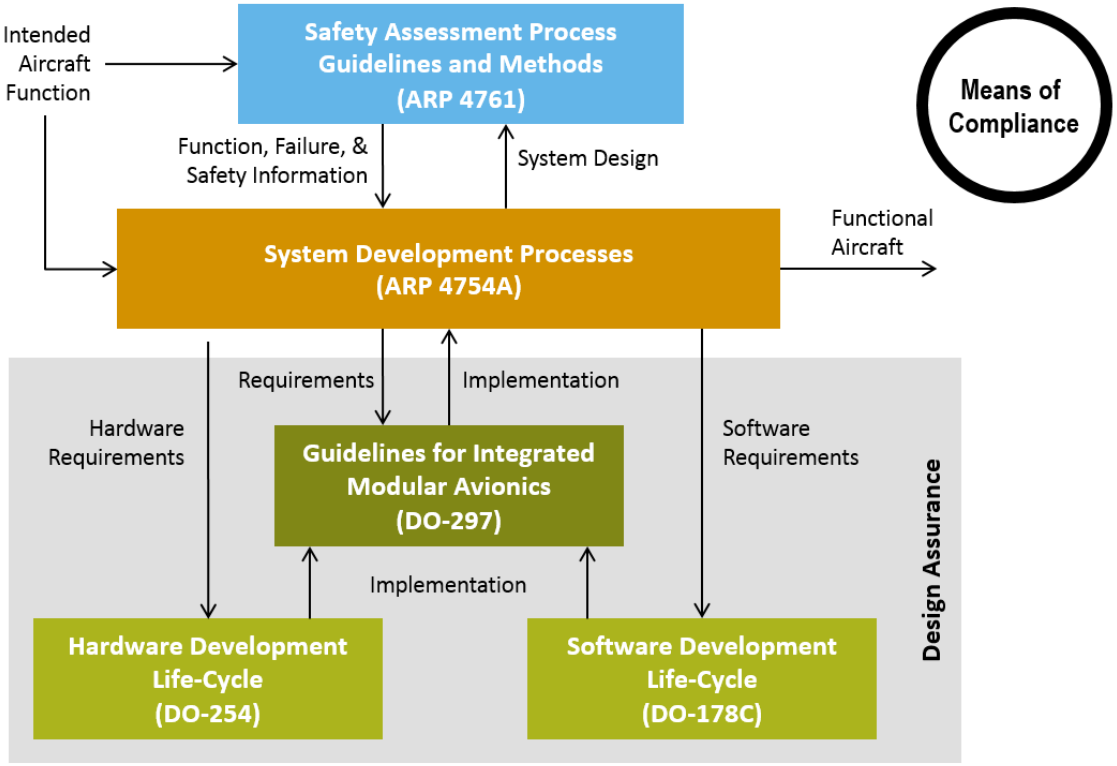

Figure 1 - Civilian Avionics Certification Overview

DO-178C defines a rigorous software development process that requires the applicant to develop requirements, design and architecture, source code, object code, and test cases and procedures. These artifacts must satisfy a set of objectives related to their accuracy, traceability, and verification. Many of these objectives have been evaluated through a process of manual review; software stakeholders

visually inspect the artifacts to establish that the requisite DO-178C objectives were met. Increasing software complexity has led to the use of software tools to assist in satisfying these objectives. However, when using a tool to satisfy objectives, it must be established that the tool itself is trustworthy. This can be established by manually reviewing the tool's output, which might be feasible for a test case generation tool. If the output of the tool is not independently verified, then the tool must be ***qualified***. Qualification is the process by which a software tool is deemed trustworthy. Guidance for qualifying tools is provided in *DO-330: Software Tool Qualification Considerations* [5]. In a process similar to DO-178C, this document outlines a set of activities for establishing that a software tool can be trusted to carry out its mission in the software development process.

As software complexity has grown, the avionics community has also turned to emergent technologies to mitigate the high cost of modern software development. In fact, the release of DO-178C also provided guidance on the use of these emergent technologies in DO-178C. This guidance comes in the form of technology supplements. These supplements interpret the guidance in DO-178C through the lens of the prospective technology. These supplements address model based development, object oriented technology, and formal methods. This project focuses on the use of formal methods tools. Guidance for their usage is provided in *DO-333: Formal Methods Supplement to DO-178C and DO-278A* [6].

Prior NASA-sponsored work entitled *Formal Methods Case Studies for DO-333* [7] described in detail how one might use formal methods tools to satisfy DO-178C objectives. The focus of the current project is to extend that work by thoroughly exploring the issues surrounding the qualification of formal methods tools. It seeks to interpret the guidance of DO-333 and DO-330 and provide critical feedback to the avionics community on how to qualify formal methods tools.

2.1 Intended Reader

This report is intended for

1. software developers and prospective certification applicants who may wish to qualify formal methods tools in a DO-178C software development process,
2. formal methods tool developers who wish build robust formal methods tools to be used in the avionics industry, and
3. practicing certification experts who will be expected to evaluate formal methods based certifications.

It is hoped that the concepts and experiences documented in this report will provide insight into the DO-330 and DO-333 guidance.

It is recommended that the reader has a general familiarity with the DO-178C software development process, including the required artifacts to be developed. It is also recommended that the reader is familiar with at least one of the following classes of formal methods tools: model checking, theorem proving, or abstract interpretation.

2.2 Overview of Project Activities

The purpose of this project is to thoroughly investigate the qualification of formal methods tools. The first activity focused on organizing an event to engage the certification and formal methods communities to discuss qualification of formal methods tools. This event, a Dagstuhl Seminar, focused on providing a broad overview of the certification processes to the formal methods community. Further,

participants provided experience reports on past formal methods qualifications within various industries, such as rail, nuclear, and automotive industries. Lastly, the entire group discussed non-standard approaches to qualification, including the use of qualified tools to check the results of unqualified tools.

Next, the project addressed the general issue of trust in formal methods tools. The *Tool Assurance Study* provides a broad overview of potential trust issues in formal methods tools. It identifies issues with formal methods tools in general and then identifies issues in trust specific to each class of formal methods tool. This document will be useful to prospective users of formal methods tools to help understand the roles of the tool developer and tool user in obtaining trustworthy results from a formal methods tool.

The remainder of work on the project focuses on the interpretation of the DO-330 qualification guidance as it applies to formal methods tools. First, the *Mitigation Strategies* activity performed a review of the tool qualification objectives anticipated for formal methods tools. This is covered in detail in Section 3.3.1, but broadly stated, it is anticipated that formal methods tools are much more likely to be used for verification rather than software development. As such, the requirements for their qualification are lower than they would be for tools used to produce airborne software. The Mitigation Strategies task performs an overview of each qualification objective and highlights concerns regarding unique challenges to meeting that objective for a formal methods tool. Additionally, for some objectives, the authors provide recommended practices for qualifying formal methods tools. This activity and the associated document provide an overview of the qualification of formal methods tools and can be used as a primer for those seeking to qualify tools within their own DO-178C software development process.

Next, three representative tool qualification plans were produced. These plans outline the required activities necessary to qualify a representative tool for each class of formal methods. The selected tools are Kind 2 [8] [9] (model checking), PVS [10] (theorem proving), and the Frama-C [11] Value Analysis Plug-in [12] (abstract interpretation). These tool plans can serve as case studies. They walk through the qualification activities and provide discussion on areas of particular interest for each specific tool. Further, they provide labor estimates for fully qualifying their respective tools. These case studies will provide useful information to tool users, tool developers, and certification experts, on interesting aspects of qualifying formal methods tools.

Finally, two competing approaches for qualifying an open source model checker, Kind 2, are provided. The first approach is a straightforward qualification of Kind 2 itself, while the second approach is based on using a proof-checking tool to evaluate the results of the outputs produced by Kind 2. On the project, Kind 2 was modified to emit a proof certificate that captures the deductive steps used to determine whether or not a property is valid. This proof certificate can be checked by a separate tool to validate Kind 2's results. The proof checking tool is named Check-It, and it is an instantiation of the Logical Framework with Side Conditions (LFSC) proof checking framework. Check-It can examine the proof certificate emitted by Kind 2, and evaluate whether Kind 2's results are trustworthy. Essentially, Check-It reviews Kind 2's outputs, much like a human might review the output of a source code generator. These two competing approaches are provided so that they can be compared, with conclusions drawn about the efficacy of the proof-based approach within the DO-178C methodology.

3 Background

This section provides background information on the organization and content within the DO-178C family of documents. It also introduces the use of software tools and the process for establishing trust in them, termed qualification.

3.1 Aircraft Certification

Certification is the process for establishing that an aircraft can be trusted to operate safely in the National Airspace System. Certification is governed in the U.S. by the Federal Aviation Administration (FAA). Technically speaking, certification applies to the entire aircraft, its subsystems, and its specific configuration. This means that certification is not really modular. A software function or component cannot be modified without revisiting its impact on the entire aircraft, and considering whether or not that change is appropriate within the context of an aircraft configuration. This is a critical concept and it is pervasive throughout the civil aviation certification guidance standards.

3.2 DO-178C and Related Documents

DO-178C provides guidance for the software aspects of aircraft certification. Restated, if an aircraft component contains software then the certification requirements pertaining to the software can be satisfied by following the guidance provided in DO-178C. Similarly, *DO-278A: Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems* [13] provides guidance for software development processes for software aspects of certification for ground-based systems, such as air traffic control systems.

In practice, DO-178C describes a software development process that when followed, will produce certifiable software. DO-178C identifies a set of planning, development, and verification objectives as part of its processes. It describes a series of software development artifacts including high-level requirements, low-level requirements (consisting of software design and architecture), source code, and object code. Each artifact must then be shown to satisfy a series of objectives related to the correctness of the artifact, suitability with respect to the hardware that it is executed on, and compliance with and traceability to previous artifacts. Finally, it requires the use of testing to verify that the object code complies with all of the requirements. Figure 2 provides an overview of the artifacts, development

objectives (lines colored in black), and verification objectives for each artifact (blue, green, purple, red, and orange lines).

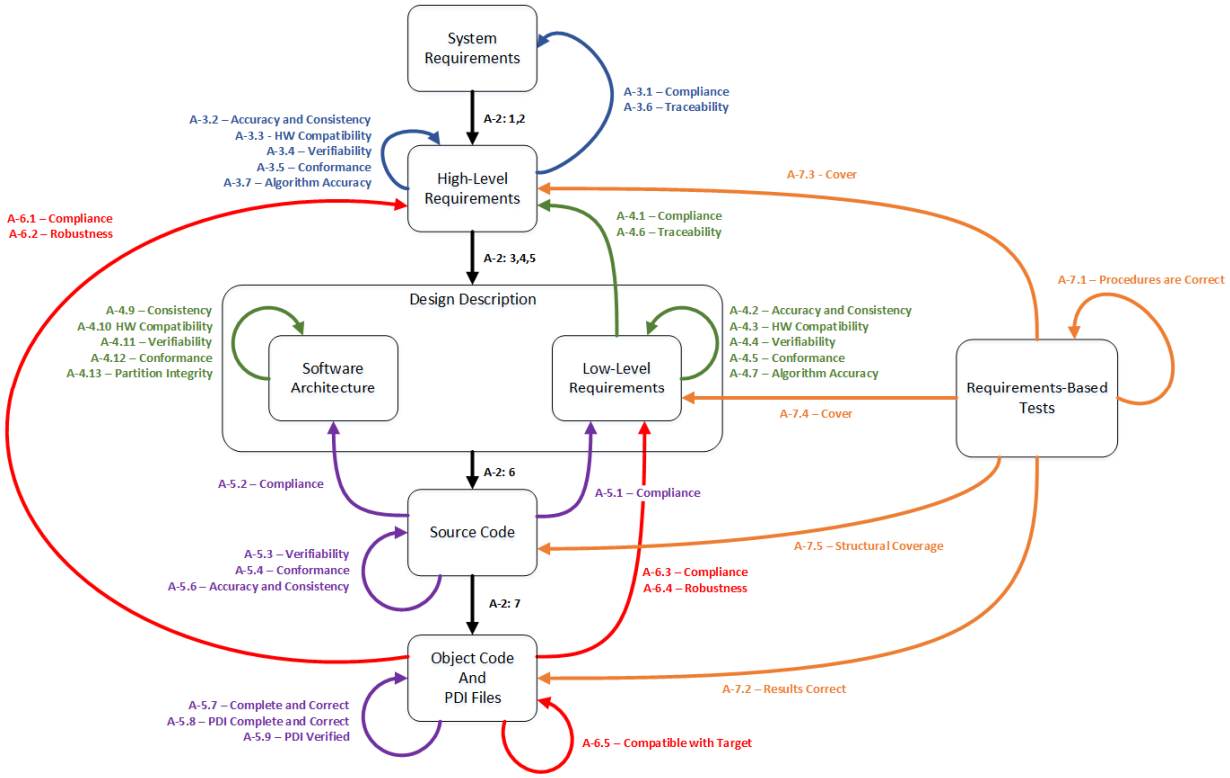

Figure 2 - DO-178C Overview

Figure 2 outlines the objectives required for the most critical avionics software, Level A. As the software criticality level decreases, so do the required objectives to be satisfied. DO-178C identifies five software levels based on the impact of a software failure on the overall aircraft function. Table 1 defines the five levels of software in DO-178C and the number of objectives associated with each level.

Software Level	Applicable Objectives	Impact on Aircraft Operations
A	71	A failure in the software could cause or contribute to a system failure resulting in multiple fatalities and/or loss of the aircraft.
B	69	A failure in the software could cause or contribute to a system failure that causes: • Large reduction in safety margins or functional capabilities • Physical distress and excessive workload such that the crew cannot be relied upon to perform their tasks accurately or completely • Serious or fatal injury to a relatively small number of occupants other than the flight crew.
C	62	A failure in the software could cause or contribute to a system failure that causes a reduction in the capability of the airplane or ability of the crew to cope with adverse operating conditions. This includes: • Significant reduction in safety margins or functional capabilities • Significant increase in crew workloads or discomfort to flight crew • Physical distress of passengers or crew, possibly resulting in injuries
D	26	A failure in the software could cause or contribute to a system failure that does not significantly reduce airplane safety and which involves crew actions that are well within their capabilities. This includes: • Slight reduction in safety margins or functional capabilities • Slight increase in crew workload • Some physical discomfort to passengers
E	0	No impact.

Table 1 - DO-178C Software Levels

Depending on the associated software level, the process can be very rigorous (Level A) or non-existent (Level E). As aircraft have increasingly relied on software to perform more complex tasks, the costs associated with certifying software have risen. As a result, the community has turned towards several emergent technologies, implemented by software tools, to mitigate complexity and cost increases associated with certification. These emergent technologies are addressed in a set of supplements to DO-178C. The first, *DO-331: Model-Based Development and Verification Supplement to DO-178C and DO-278A* [14] provides guidance on the usage of model-based development tools. *DO-332: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A* [15] provides guidance on the usage of object oriented programming and related techniques. *DO-333: Formal Methods Supplement to DO-178C and DO-278A* [6] addresses the use of formal methods.

These technologies are implemented in the form of software tools that help perform various aspects of software development and verification. When a software tool is used to satisfy objectives of DO-178C, then it must be qualified. This project is focused on the qualification process. The process for qualifying tools is addressed in *DO-330: Software Tool Qualification Considerations* [5].

Figure 3 shows the DO-178C family of documents.

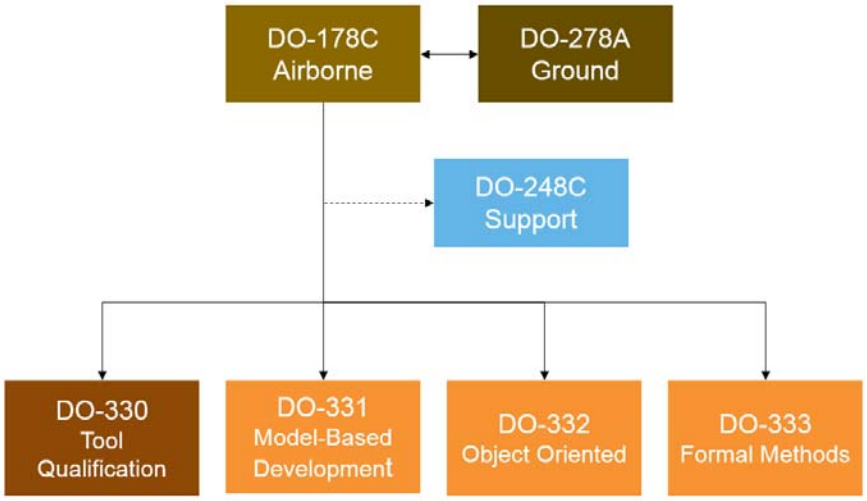

Figure 3 - DO-178C Document Tree

3.3 The Role of Tools and Qualification

Tools are ubiquitous in modern avionics software development efforts. Requirements management tools such as IBM® Rational® DOORS®, model-based development environments such as Simulink®, and compilers such as GCC have all been successfully used in DO-178C compliant software development processes. When a tool is used to satisfy DO-178C certification objectives its outputs must either be manually reviewed or the tool itself must undergo a process to establish its suitability for meeting the claimed objectives. This process is known as qualification. Formal methods tools provide the unique capability to be able to reason about airborne software. In most situations, a formal methods tool can provide a much stronger argument for the correctness of software. As for other tools, when a formal methods tool is used, its outputs must either be reviewed to ensure its results are correct and trustworthy, or the tool itself must be qualified. However, many formal methods tools either do not provide any reviewable output at all, or provide complicated output that cannot be reliably reviewed by a human being. As such, qualification of a tool is required.

DO-330 outlines a process by which one can evaluate a tool's suitability for satisfying DO-178C objectives it claims to automate, reduce, or eliminate. This qualification process is similar to the certification process defined in DO-178C. Qualification amounts to accomplishing a set of activities with corresponding objectives to:

- Identify which DO-178C objectives the tool is being used to automate, reduce, or eliminate
- Specify which functions of the tool are being relied upon
- Create a set of test cases or analyses that demonstrate that the tool performs correctly.

As in certification, there are varying levels of rigor associated with tool qualification. The rigor of a tool's qualification efforts is dictated by the tool's impact on the software development process.

3.3.1 Determining the Tool Qualification Level

The tool qualification level (TQL), similar to the software level in DO-178C, defines the level of rigor required by the qualification process. Tool Qualification Level (TQL) 1 is the most rigorous, while TQL-5 is

the least rigorous. Three criteria are defined in DO-178C to determine the impact of an error in the tool on the overall software development process. Table 2 identifies the three criteria for a software tool.

Tool Criterion	Impact of Tool
Criterion 1	A tool whose output is part of the airborne software and thus could insert an error.
Criterion 2	A tool that automates verification processes and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of: • Verification processes other than those automated by the tool. • Development processes that could have an impact on the airborne software.
Criterion 3	A tool that, within the scope of its intended use, could fail to detect an error.

Table 2 - Defining a Tool's Criterion

Given the nature of formal methods tools, it is expected that they will be used in a verification context within the DO-178C software development process. Thus, it is expected that formal methods tools will fall into either Criterion 2 or Criterion 3. That is, they will not be used to produce airborne software, but used to verify that the airborne software is correct. The distinction between Criterion 2 and Criterion 3 depends on which processes the tool is being used to automate, reduce, or eliminate. For example, if an abstract interpretation tool concludes that division-by-zero cannot occur in a piece of airborne software, and this is used to partially satisfy DO-178C objectives related to the accuracy and consistency of the source code artifact (Objective A-5.6), then the tool is Criterion 3. However, if those results are then used as a justification to reduce robustness testing related to division-by-zero in the object code testing (Objectives A-6.2 and A-6.4), the tool becomes a Criterion 2 tool. An official rule of thumb is that when a tool addresses objectives from multiple tables of DO-178C, it is potentially a Criterion 2 tool.

The TQL is determined by using DO-178C Table 12-1, which is reproduced below in Table 3, and cross-referencing the identified Tool Criterion, with the DO-178C software level to which the tool is being applied.

Software Level	Criterion		
	1	**2**	**3**
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Table 3 - Tool Qualification Level Determination

For example, if one wanted to qualify a compiler for use on level C software, one would have to determine its tool criterion (since it produces airborne software, it is criterion 1) and then use Table 3 to determine that it would be a TQL-3 qualification effort. Since it is expected that formal methods tools will only be used as criterion 2 or 3 tools, it is anticipated that they will need to be qualified as either TQL-4 or TQL-5 tools.

3.3.2 DO-330 and Tool Qualification Objectives

Once the required TQL is determined, one can then use the document to identify the relevant tool qualification objectives. DO-330 is organized such that objectives relating to a specific portion of the tool's operational and development processes are grouped together in tables. For example, Table T-0 captures objectives related how the tool is used within the context of the larger software development effort, also known as the tool's operational processes. Tables T-1 through T-7 address tool planning, design, development, integration, and verification. Tables T-8, T-9, and T-10 address configuration management of the tool, the role of quality assurance, and identification of the tool in the larger aircraft certification effort respectively.

Shown below in Table 4 are the set of objectives required for TQL-4 and TQL-5 qualifications, which are the anticipated levels for formal methods tools.

DO-330 Table	TQL-4 (24)	TQL-5 (14)
T-0: Tool Operational Processes	4	1,2,3,5,6,7
T-1: Tool Planning Processes	1,4	
T-2: Tool Development Processes	1,2,3,7,8	
T-3: Verification of Outputs of Tool Requirements Process	1,2,3,4,5,6,8,9	
T-4: Verification of Outputs of Tool Design Process	10	
T-5: Verification of Outputs of Tool Coding & Integration Process		
T-6: Testing of Output of Integration Process	1,2	
T-7: Verification of Outputs of Tool Testing	2,3	
T-8: Tool Configuration Management	2,3,5	1,4
T-9: Tool Quality Assurance Process		2,5
T-10: Tool Qualification Liaison Process		1,2,3,4

Table 4 - TQL-4 and TQL-5 Qualification Objectives

The qualification objectives for a particular TQL are cumulative. It includes all of the objectives for the selected TQL and the objectives for all the levels below it. For example, in a TQL-4 qualification, the applicant is expected to satisfy the stated 24 objectives for as TQL-4 qualification, and also the 14 objectives for TQL-5, for a total of 38.

From Table 4 one can observe that a TQL-5 qualification is relatively simple. The applicant must identify how the tool is used within the software development process, identify the tool's operational requirements, and demonstrate that the tool satisfies those requirements. This is an important point. Qualifications, much like certifications, are performed in the context of the aircraft development effort in which they are applied. This means that certain tool functions may not be utilized or addressed in a qualification. For example, qualification of a code generator may only consider a subset of the legal input for the source language; qualification of a model checker may only cover variables of primitive input types and choose to ignore composite types such as arrays, records, and tuple types. The context of the tool's usage, and verification that the tool operates correctly within that context, is always required. In the case of TQL-5 qualifications, it is all that is required, in addition to the activities of tables T-8, T-9, T-10.

TQL-4 qualifications represent a significant increase in effort because they require the applicant to produce artifacts from the software development process of the tool itself. This may include tool

planning documentation, tool requirements, tool test cases and procedures, and tool testing results. The result is that TQL-4 qualifications require insight into how the tool was built. If a tool was built by a third party, TQL-4 and above qualifications are not achievable without artifacts provided by the tool developer. Depending on the history of the tool, this may be difficult, or impossible, to obtain.

Once the proper TQL is selected and the objectives have been identified, qualification is simply a matter of demonstrating that each objective is satisfied. Part of this project's goals is to enable the reader to successfully identify a tool's criterion, select a TQL, and satisfy the anticipated tool qualification objectives for a formal methods tool.

4 Dagstuhl Seminar on Qualification of Formal Methods Tools

Schloss Dagstuhl is a venue for computer scientists to meet and discuss interesting problems relevant to academia and industry. This project helped to organize a Dagstuhl seminar to discuss the problems relevant to the qualification of formal methods tools. Attendees included both developers of formal methods tools and experienced practitioners of the DO-178C development process.

4.1 Scope and Purpose

The seminar, held April 26-29, 2015 in Dagstuhl, Germany, covered the following topics:

- Provided a broad overview of the DO-178C and DO-330 tool qualification guidance for the benefit of formal methods tool users and developers in attendance,
- Highlighted more general issues relating to the trust of formal methods tools,
- Presented a series of experience reports on past formal method tool qualifications in the avionics, rail, nuclear, and automotive industries,
- Presented and discussed the similarities and differences in tool qualification standards across industries,
- Identified and discussed nontraditional approaches to tool qualification, for example, the use of a qualified tool to check the results of an unqualified tool,
- Discussed several large formal methods projects, including the CompCert verified compiler, the verification of the seL4 microkernel, and the obstacles to tool qualification.

4.2 Seminar Activities

Due to the diverse background and job functions of the attendees, the first day of the seminar was geared to provide a detailed introduction to qualification under the guidance provided by DO-178C and DO-330. Darren Cofer gave a presentation entitled *Are you Qualified for this Position?* that provided background in the specifics of the qualification process including how to determine the need for tool qualification, how to determine the tool criterion for a given tool and usage, and how to select the appropriate tool qualification level. These concepts were reinforced by a presentation by Lucas Wagner that illustrated a past qualification effort for a test case generation tool. Finally, Konrad Slind introduced the notion of the trust proposition and addressed the general concerns of trusting a formal methods tool.

The second day of the seminar focused on experience reports in the qualification of formal methods tools in the avionics industry, and other industries such as rail and nuclear. Matteo Bordin (*Qualification of Formal Methods Tools and Tool Qualification with Formal Methods*) described an effort at AdaCore to qualify an abstract interpretation tool (CodePeer) and a formal verification tool (SPARK) for DO-178 development efforts. Remi Delmas presented work completed in the rail domain to qualify SAT-based tools (*Sharing experience on SAT-based formal verification toolchain qualification in the railway domain*). Werner Schuetz also discussed work done to qualify tools under the railway standard EN50128, in his talk entitled *Tool Qualification in the Railway Domain*. Alain Mebsout presented work performed on this project in his talk entitled *Certificates for the Qualification of the Model Checker Kind 2*. Mark Lawford presented his work on the use of PVS to verify a nuclear shutdown system, entitled *Qualification of PVS for Systematic Design Verification of a Nuclear Shutdown System*. Markus Pister talked about the tool qualification strategy employed by AbsInt, developers of abstract interpretation based tools for

reasoning about source code, stack usage of a program, and worst-case execution time. This talk was titled *Tool Qualification Strategy for Abstract Interpretation based static analysis tools*.

There was also significant discussion surrounding a presentation by Xavier LeRoy on the CompCert [16] compiler. CompCert produces a proof of equivalence between C source code and the resulting executable object code using the Coq [17] theorem prover. The participants struggled with what certification credit might be obtained through the use of this compiler and what sort of qualification might be required. Part of the problem is that DO-178C does not include certification objectives related to code compilation since it is focused on testing of the executable object code. Therefore there is no real need for qualification of a compiler.

On the third day a presentation was given by June Andronick, employee at NICTA, entitled *Please check my 500k LOC of Isabelle*. This talk focused on work completed to verify the seL4 [18] microkernel using the Isabelle HOL [19] theorem prover. This was a major focus of discussion as it was unclear how to leverage the work done by NICTA/Data61 to verify the kernel and qualify the tools used to perform the verification. This presentation illustrated the fact that is very difficult to add certification (and qualification) considerations into software and tools after the fact. In general, these considerations need to be addressed during the design of the software or tool.

4.3 Conclusions

Overall the seminar was a success. It brought together prominent researchers from the formal methods community and knowledgeable practitioners of DO-178C development processes to engage in meaningful discussions of the specifics of qualification as required by DO-330. Of particular value to this effort was engaging in the discussion with DO-178C/DO-330 experts to understand some of the finer points of tool qualification.

The seminar report can be found at http://www.dagstuhl.de/15182.

5 Tool Assurance Study

Formal analysis tools are already used throughout industry as bug-finding tools. For example, Polyspace Bug Finder™ [20], a tool based on abstract interpretation, is used in the avionics and automotive industries to identify potential run-time errors in source code artifacts. Another tool, Simulink Design Verifier™ [21], which is based on model checking, can be used to identify errors in Simulink and Stateflow models.

If a formal methods tool is used only for bug detection, it does not need to be qualified. This is because it is not being used to prove the correctness of an artifact. Instead, it is only used to improve the developer's confidence by identifying defects or suggesting the artifact is free from defects. This project addresses the usage of formal methods tools to obtain certification credit. As such, certifications based on formal methods are relying on the tools to provide accurate and valid results from reasoning. While there is a process for establishing trust in a tool in the avionics industry, namely, qualification, there is a related but more general question of whether or not a tool and its results can be trusted. Qualification is an indication of the trustworthiness of a tool, but it is more pragmatic than the general question of trust. For a formal methods tool, the answer to this question often relies both on actions by the developer and the user.

The Tool Assurance Study activity is focused on answering the general question of whether a formal methods tool can be trusted. It also provides a systematic mechanism for evaluating whether or not a tool and the results it produces are trustworthy.

5.1 Scope and Purpose

The purpose of the Tool Assurance Study is to provide a systematic approach to evaluating the trustworthiness of formal methods tools. In addition, it aims to examine past and present trust issues inherent in existing tools to provide insight into how the issues manifest themselves.

The study focuses on formal methods tools and then examines specific instances of the three main classes of formal methods tools: model checking, theorem proving, and abstract interpretation. The study does not concern the usage of SMT solvers as standalone tools. However, the use of such tools is considered within the context of a theorem proving or model checking tool, for which they are frequently packaged.

This study does not address issues related to the trust of tools in general. For example, it does not address whether or not the compiler produces a trusted binary. Those issues are addressed in the field of information security and as such, are not addressed here.

5.2 Key Findings

The Tool Assurance Study defines the notion of the trust proposition. The trust proposition is the collection of evidence required to demonstrate that a formal methods tool provides trustworthy results. There are four characteristics of formal methods tools that contribute to the trustworthiness of their results. These characteristics are:

- the theoretical basis for the analysis approach,
- the accuracy and correctness of the implementation
- the accuracy of the artifact formalization

- the accuracy of the specification

The first characteristic refers to the theoretical foundations on which the formal analysis is based upon. For example, for a model checking tool, such as Kind 2, it may reference the scientific literature to establish the theoretical basis for bounded model checking [22], k-induction [23], and property directed reachability [24] [25]. The trust basis for a formal methods tool's theoretical approach is built upon peer reviewed publication describing the approach.

The second characteristic refers to the correctness of the implementation of the tool's theoretical approach. This is focused on the details of building the tool that can contribute to the tool providing incorrect results. Some factors that might introduce errors in the implementation are incorrect library usage, translations that do not preserve semantics, and the correctness of parallelization algorithms, to name a few.

While the theoretical approach and accuracy of the tool's implementation are attributed to the tool's developer, the remaining two characteristics are attributed to the tool user. The next concern is whether or not the artifact being analyzed, the formal model, is a correct and faithful representation of the problem being analyzed. Often tools are used to translate from one format that is useful in a development capacity, to another format that is useful in a verification capacity. When such tools are used one must be sure that the semantics of the original artifact are preserved in the verification artifact. In other situations, a verification artifact might be manually created. In this case, it is important for the formalized artifact to be thoroughly reviewed to ensure it is a correct and faithful representation of the artifact it represents.

In addition to incorrect formalizations, it is also important to address limitations of the formalization. For example, Kind 2 models integers and operations over them using infinite precision variables. This means that the overflow and underflow behaviors associated with modular integer arithmetic are not modeled or analyzed using the tool. If overflow and underflow are a concern the user must either update the formal artifact to represent modular integer arithmetic accurately, or it must be identified as a gap in the formal analysis.

Finally, we address the correctness of the formal specification or property that is being used to verify the artifact of interest. Given the complicated nature of the logics often implemented in formal methods tools it can be difficult to understand when a property is incorrect, especially with respect to the boundary conditions of the property. This responsibility falls on the user. Some tools, such as abstract interpretation tools, emit properties based on the structure of the model. For example, it may emit a property that the divisor of every division operator is not equal to zero, to identify potential runtime errors. The responsibility for correctly generating these types of specifications falls on the tool developer.

The study addresses each class of formal methods tool directly and elaborates on the trust proposition for each. The mechanisms used for each tool may amplify or attenuate concerns in each. For example, for model checking tools, users typically generate specifications manually. In large efforts requiring accurate results, it is wise to engage in a peer review process to ensure these are correct. However, in abstract interpretation tools, specifications may be generated automatically by the tool, and thus the concern for human introduced errors is less.

5.3 Conclusions

This activity presented a systematic approach for identifying the various components of trust in formal methods tools, highlighting the various aspects of formal methods tools that could contribute to the production of incorrect results. Incorrect results can be produced by errors in the mathematical basis of the tool, errors in the tool implementation, or by misunderstandings on the part of the user when creating a formalization or specification. In the report, we have provided examples of issues in existing tools to illustrate what can go wrong.

The conclusion from this activity is that formal methods tools provide a high degree of assurance. However, it can be difficult to ensure they are designed, implemented, and used correctly. From the perspective of certification, therefore, it is important to know the tool's capabilities and potential defects so that they may be mitigated. The Tool Assurance Study confirms both the need for a firm understanding of the logical and algorithmic basis of the tool and, of equal importance, the need for good software engineering practices in the production and ongoing development and maintenance of the tool.

The full report is available as a companion document, entitled Tool Assurance Study.

6 Mitigation Strategies

Qualification of tools is not a new concept. DO-178B introduced the process of tool qualification. It made the distinction between development tools and verification tools. The former requiring the rigors of a DO-178B-like process for the tool itself, and the latter requiring a process similar to that of DO-330 TQL-5 type qualifications. The revision from DO-178B to DO-178C introduced more rigorous guidance for tool qualification. It created a stand-alone document, DO-330, to defining the process of tool qualification, with the intention that this document could be used in other industries.

In addition to improved guidance on tool qualification, guidance was provided to identify acceptable usage of formal methods tools. This new guidance presents an opportunity for the avionics community to use formal methods tools to reason about software development artifacts and satisfy certification objectives based on those results.

Qualification of tools is well understood. However, qualification of formal methods tools, and potential difficulties in meeting the qualification objectives for them, is relatively unstudied. This activity seeks to improve understanding of the potential qualification difficulties present in the guidance for formal methods tools.

6.1 Scope and Purpose

The purpose of this activity is to review the tool qualification objectives within DO-330 to identify aspects of them that might be difficult to satisfy for formal methods tools. When a difficulty is identified, potential mitigation strategies are identified. In addition to identifying issues, it also recommends practices that the authors feel would be helpful for formal methods tool users who are new to the DO-178C certification and DO-330 qualification processes.

As explained in Section 3.3, the rigor of qualification is determined by the tool's impact on the overall software development process. The impact of the tool drives the selection of the tool's criterion, as illustrated in DO-178C Section 12.2.2. Table 2 from Section 3.3.1 provides definitions for the three potential tool criteria.

It is anticipated that formal methods tools will be used in a verification capacity; that is, they will not be used to generate airborne software. As such, they are likely to be identified as Criterion 2 or 3 tools. With this information, one can use DO-178C Table 12-1 (reproduced in section 3.3.1, Table 3) to identify the potential tool qualification levels, often referred to simply by the acronym TQL. From inspection, one can see that TQL-4 and TQL-5 qualifications will be required for formal methods based verification tools.

This activity performs a review of all objectives necessary for a TQL-4 and TQL-5 qualification effort. This is the highest level of rigor expected of formal methods tools. The relevant objectives are identified in Table 4, of Section 3.3.2.

6.2 Key Findings

One major and several minor issues were discovered as part of this activity. These issues are summarized in Table 5, complete with page numbers to find the discussion regarding the issue in the Mitigation Strategies document.

Issue Description	Severity
TQL-4 qualifications require artifacts from the tool developer (pg. 5)	Major
Interactive tool features may needlessly complicate qualification (pg. 7)	Minor
Termination of analysis may not be guaranteed (pg. 7)	Minor
Establishing well-formed input through testing (pg. 8)	Minor
Verification of tool testing results manually may be impractical/untrustworthy when outputs are voluminous (pg. 13)	Minor

Table 5 - Issues Identified in the Mitigation Strategies Task

Discussion of the minor issues is provided within the Mitigation Strategies document. The major issue, which relates to the ability to perform TQL-4 qualifications for tools without developer support, is addressed below.

A TQL-4 qualification requires the tool developer to provide documentation regarding the tool development process. While this issue is not specific to formal methods tools, it is potentially a looming factor for many formal methods tools, especially those arising from the research community. If the tool developer has not produced the requisite artifacts, it may not be possible to pursue a TQL-4 qualification at all.

If the tool development artifacts required by TQL-4 cannot be provided or somehow produced retroactively, there is really only one strategy to mitigate this issue. The tool user must reduce the number of certification objectives claimed by the tool, such that it avoids classification as a Criterion 2 tool, and the tool can be qualified per TQL-5.

Lastly, we identify one open question. DO-330 Objective T-2.3 requires the applicant to identify and develop the tool's architecture. This includes identifying relevant features, protection mechanisms, and external libraries used, among other things. When a tool, such as an SMT solver, is used as a library within a formal methods tool to perform a most of the reasoning (it is estimated that an SMT solver performs 75% of the reasoning in Kind 2) is simply documenting the use of such a library adequate? Should there be additional verification associated with the library itself?

Discussions with certification experts indicated that testing the tool in its operational environment is satisfactory for establishing that the tool, and its internal components, are suitable for meeting the tool's objectives. It is simply highlighted here so that the community may be aware of it since it appears to be a unique feature of many formal methods tools.

6.3 Conclusions

The conclusion drawn from this activity is that the tool qualification guidance in DO-330 anticipates the use and qualification of formal methods tools. There is one relatively major issue regarding the need for tool developer support for TQL-4 qualifications, but beyond that, there are no obstacles that would prevent one from successfully qualifying a formal methods tool.

Several minor issues were also identified in this activity, but the Mitigation Strategies document explains each and identifies strategies for managing them. This activity and produced document can serve as a primer for those pursuing qualifications for formal methods tools.

The full report is available as a companion document, entitled Mitigation Strategies.

7 Tool Qualification Plans and Estimates

In 2013 the FAA recognized DO-178C, thereby updating the guidance on the software aspects of certification from the former document, DO-178B. The DO-178B standard was in place since its introduction in 1992. One of the goals of the committee that created DO-178C was to preserving the core aspects of DO-178B as much as possible while still incorporating guidance for new technologies. As such, the DO-178C document is a relatively minor revision of DO-178B. Revisions to improve clarity and comprehension were made, but the style, organization, and content of the document were not significantly changed. The major change introduced by this revision was the introduction of the three technology supplements, DO-331, DO-332, and DO-333, and the introduction of a standalone document that provides guidance on tool qualification, DO-330.

This new guidance presents the possibilities of using formal methods for certification credit, which previously was only mentioned in DO-178B as an "alternative method," and so was viewed as risky. Despite this newly released guidance, there is still risk associated with the use of new technologies and tools. It was determined that a set of case studies that examined the usage and qualification of formal methods tools would be beneficial for the avionics community and help reduce the uncertainty associated with the use of formal methods tools for certification. The *Formal Methods Case Studies for DO-333* produced case studies on how to use formal methods tools to satisfy certification objectives. This activity provides complementary case studies on how to qualify formal methods tools. Together these examples can serve as a guide to both tool users and certification experts in understanding, concretely, the details of using formal methods tools for certification and qualification.

7.1 Scope and Purpose

The purpose of this activity was to develop case study tool qualification plans for each class of formal methods tool. These plans identify the activities necessary to qualify a representative tool from each class of formal methods tool, at a TQL-5 level of rigor, and demonstrate how to satisfy each objective. The selected tools are:

- Kind2, a model checker,
- PVS, an interactive theorem prover,
- Frama-C Value Analysis plug-in, an abstract interpretation tool

These plans take the information learned as part of the Mitigation Strategies activity (see Section 6) and apply it to develop a concrete case study for each of the listed tools. In addition, each plan provides a labor estimate to help prospective tool users understand the level of effort required for qualification. In addition, they provide discussion and feedback to the reader on the more subtle, confusing aspects of qualification.

Note to the reader: The Kind 2 qualification plan is not included in this report as a case study. Instead, a full qualification package for Kind 2 is provided and described in Section 8. In addition, the actual labor required to develop the full package is provided, rather than estimates.

As established in Section 3.3, formal methods tools are anticipated to be qualified to TQL-4 or TQL-5. Section 3.3.2 identifies the objectives for both levels. As explained, qualifications are cumulative, that is, the activities of a TQL-4 qualification also contain the activities of a TQL-5 qualification. Further investigation reveals that the activities of a TQL-5 qualification are the responsibility of the tool user;

they require the applicant to identify the objectives the tool is addressing in the overall certification process, identify the tool's features that are being relied upon, and develop test cases to demonstrate it performs adequately in the tool's operational environment. The TQL-4 portions of a qualification effort focus on the activities of the tool developer. For these reasons, the decision was made to tailor these case studies as TQL-5 qualification plans.

7.2 Key Findings

The Kind 2 plan focuses on the qualification of Kind 2 to satisfy objectives from DO-178C Table A.4, which are verification objectives for the design artifacts. Kind 2 supports multiple Satisfiability Modulo Theory (SMT) solvers including CVC4, Yices, and Z3. The use of a specific solver induces different behaviors in Kind 2 itself. The provided qualification package focuses on the use of Z3. As a result, behaviors such as nonlinear arithmetic are allowed. Other solvers do not allow such constructs. The point is that the choice of solver significantly alters the behavior of the tool. From the perspective of qualification, Kind 2 can potentially be qualified as three different tools; each utilizing a different SMT solver. This very fact emphasizes the importance of identifying the tool's operational context and which features are being relied upon to satisfy certification objectives.

The PVS plan focuses on the qualification of the tool in its capacity to replay proofs. These proofs are used to satisfy objectives from DO-178C Table A.3, which contains verification objectives related to the high-level software requirements. PVS contains many functions and features; among those are the interactive capabilities for users to incrementally build proofs of theorems in their formal artifacts. However, from the certification perspective, it is irrelevant how the user developed a proof. Instead, the most relevant function of the tool is the ability to demonstrate that a proof is correct. As a result, the qualification approach for PVS focuses on the batch mode of operation. The thought is the user can develop their proof of correctness as part of the development process. This proof can then be replayed by the tool in batch operation to demonstrate satisfaction of certification objectives. The takeaway is that many functions of a tool, such as the generation of counterexamples or interactive proof development capabilities, may improve the user's experience, but may not actually support the certification objectives the tool claims. It is in the best interest of the tool user pursuing qualification to identify capabilities of the tool that support the certification objectives and only qualify those capabilities. The remaining capabilities, however useful, only serve to increase the cost and complexity of the qualification effort.

Finally, the plan for the Frama-C value analysis plug-in examines the ability of the tool to identify run-time errors in source code, which support the satisfaction of objectives from DO-178C Table A.5. There are no unique aspects of formal methods at play in this case study. In fact, a process for satisfying the objectives of Table A.5 within the industry will likely include several source code analyzers including linting tools, style checkers, abstract interpreters, and perhaps even, model checkers such as CBMC [26].

7.3 Conclusions

The outputs of this activity are three tool qualification plans that identify the qualification objectives and provide supporting information to satisfy those objectives, for a TQL-5 qualification of formal methods tools. A plan is provided a specific instance of each class of formal methods tool. A full qualification package, complete with tool operational requirements and test cases, is provided for the Kind 2 model checker, and further described in Section 8. For PVS and Frama-C, a theorem prover and abstract interpretation tool respectively, a tool qualification plan is provided. These plans identify all of the

necessary activities for qualification, and provide a subset of tool operational requirements and test cases, as examples for the reader.

These plans demonstrate the straightforward nature of tool qualification and illustrate how the guidance in DO-330 applies to the qualification of formal methods tools.

The full tool qualification plans are available as companion documents, entitled:

- PVS Tool Qualification Plan and Estimates
- Frama-C Value Analysis Qualification Plan and Estimates
- Kind 2 Qualification Package

8 Qualification Package of Kind 2

As mentioned in the previous section, a complete qualification package for Kind 2 was produced as part of this effort. This package expands on the tool qualification plan case studies by providing a full suite of tool operational requirements and test cases and procedures. The tool operational requirements identify which constructs of the Lustre [27] language are supported as part of the qualification, what input validations are expected to be performed, define performance requirements, and identifies the expected behaviors for checking properties within Lustre files.

It also provides a comprehensive set of review procedures to help establish that the tool conforms to its stated input restrictions, and test procedures that exercise the tool and demonstrate that it meets its stated operational requirements.

8.1 Scope and Purpose

The purpose of this qualification package was to provide a complete case study, in contrast to the PVS and Frama-C Value Analysis tool qualification plans, containing a detailed set of tool operational requirements and test procedures. It is anticipated that this qualification package contains all of the necessary information such that it could be used within an avionics certification effort. Successfully using it would require the applicant to provide detailed information to support the tool qualification objectives from Table T-8, T-9, and T-10, which are specific to an organization's configuration management, quality assurance, and certification practices respectively.

This qualification of Kind 2 is limited in its scope. It does not consider all of the functionality in the tool's current release, version 1.0.1. Instead, it selects a subset of behaviors allowed in the overall tool. The selected subset covers all of the behaviors necessary to use Kind 2 to satisfy the objectives outlined in the *Formal Methods Case Studies for DO-333* work. A comprehensive list of supported features in the qualification package is:

- Logical operations (and, or, xor, implication)
- Arithmetic expressions (+, -, *, /, div, mod)
    - Addition, subtraction, and multiplication over reals and integers
    - Integer division (div) and modulus (mod)
    - Real valued division (/)
- Relational operations (>, >=, <, <=) over arithmetic types
- Equivalence (=, <>) over all types
- If/Then/Else expressions
- Temporal operators (arrow and pre)
- Node Call expressions
- Record literals, access, and non-destructive updates
- Tuple literals, access, and non-destructive updates
- Boolean types
- Integer types
- Integer subrange types
- Real types
- Record types
- Tuple types
- Named types
- Input validation

    - type correctness of all constructs
    - detection of cyclic definitions
        - types
        - expressions
        - nodes
        - constants
    - nonlinearity detection
    - identification of invalid constants
    - identification of conflicting ids
        - constants
        - types
        - nodes
        - variables
    - identification of invalid subrange types
    - identification of multiple assignments to node variables
    - identification of attempts to assign input variables
- Identification of valid properties.
    - in the main node
    - in called nodes
- Correct handling of assertions
    - in the main node
    - in called nodes

8.2 Key Findings

The qualification testing suite identified several issues in which Kind 2 did not fully meet its tool operational requirements. These issues were related to Kind 2's ability to validate its inputs. The issues were:

- Identification of non-constant expressions for constant declarations,
- Identification of cyclic references among types and constants,
- and identification of illegal subrange types

The impact of each issue is discussed in detail within the Kind 2 qualification package. The first error has been addressed in a bug-fix by the Kind 2 developers. The remaining two errors are slated to be fixed but do not have serious consequences if they are encountered. The first error causes the analysis to hang so the user does not receive any results, much less erroneous results. The second error results in an unsatisfiable model and thus the tool outputs a warning to the user.

There is a more general concept to be addressed here. Tools with errors may be used for certification credit, provided that issues are documented, their impact noted, and a justification or mitigation approach is identified. Of course, if a tool contains so many errors that it removes all confidence in it, then it is still not suitable for use on certification efforts. When a tool contains errors it is important for the impact of those errors to be understood, and for the applicant to be able to establish that the error does not impact the correctness of the tool's analysis in the underlying certification.

The Tool Assurance Study, described in Section 5, addresses the sources of errors in formal methods tools. All tools will have errors, including formal methods tools. However, this should not preclude their

use in certification contexts. Instead, it is important to know how a tool can behave incorrectly, and document that limitation. This finding reinforces the conclusions of the Tool Assurance Study; it is important for tool developers to follow good software engineering practices if they anticipate their tools to be used in a certification context.

8.3 Conclusions

This tool qualification package provides a complete case study for qualifying an open source model checker, Kind 2. It identifies all the activities necessary to qualify Kind 2 to claim certification credit for objectives from Table A.4. This includes a complete list of tool operational requirements and test cases to exercise the tool and demonstrate those requirements are met. It is anticipated that this qualification package could be used within certification efforts claiming similar objectives to those claimed in this case study. Further, this qualification package could be used as starting point to build qualification packages that consider more complex capabilities of Kind 2, or even JKind [28].

The full report is available as a companion document, entitled Kind 2 Qualification Package.

9 Development of a Proof-Emitting Version of Kind 2

Kind 2 is a model checker for verifying *safety properties* of systems modeled using the Lustre language. Many reactive systems used in the design of safety-critical aerospace systems can be modeled using Lustre, or translated to Lustre from other languages. Like other model checkers, its success is due to the fact that this is an entirely automatic "push button" tool. One clear strength of model checkers, as opposed to proof assistants, is their ability to return precise *error traces* witnessing the violation of a given safety property. In addition to being invaluable to help identify and correct bugs, error traces also represent a checkable unsafety certificate.

In contrast, most model checkers are currently unable to return any form of corroborating evidence when they declare a safety property to be satisfied by a system under analysis. We remedy this unsatisfactory situation in Kind 2 by offering a proof production mechanism which we briefly discuss in this section and whose details can be found in Proof-Emitting Kind 2 technical report provided as a companion document.

9.1 Background

Verification approaches for software tools most often rely on traditional verification approaches, which rely heavily on testing to demonstrate correctness. The problem with test-based approaches is that they do not provide a proof of correctness; they only demonstrate the absence of errors on the identified test cases.

This section explores two alternative verification approaches based on formal methods. The first approach relies on the use of formal methods tools to verify other formal methods tool. These approaches attempt to prove the tool is correct for all possible inputs. The second approach focuses on verifying the tool's results for a single execution. The tool itself emits an artifact which is then checked by a second tool, to be correct. Each approach is explained in the following sections.

9.1.1 Verified Tools

A natural approach to the qualification of verification tools consists in proving the program (here the model checker) correct once and for all. This is possible to a large extent for programs written in programming languages with (largely automated) verification toolsets such as ESC Java 2, Frama-C, VCC, F*, *etc.* Proving full functional correctness of a model checker, however, is currently a very challenging job because these tools are often rather complex and tend to evolve quickly with the ongoing advances in the field. When feasible, one great advantage of this approach is that the performance of the model checker is minimally impacted by the verification process.

Another possibility is to prove the underlying algorithms of a model checker correct in a descriptive language of interactive proof assistants such as Coq or Isabelle, and obtain an executable program from these tools through a refinement process or code extraction mechanism. In the recent years, certified software of this category has gained interest (*e.g.*, the C compiler CompCert, or the operating system micro-kernel seL4).

While this approach may seem better at first, based on the fact that the tool is verified once and for all, it has a number of disadvantages. To start, the effort is normally enormous since there are no general frameworks for verifying modern model checkers. Moreover, any modifications to the originally verified

tool will require proofs to be re-established. In more extreme cases (*e.g.*, an in-depth modification) one may have to invest the same amount of effort as for the original correctness proof.

9.1.2 Verifying Tools

An alternative is to instrument the model checker so that it is *certifying*, *i.e.* it accompanies its safety claims with a *proof certificate*, an artifact embodying a proof of the claim. This artifact is traditionally called a *certificate*. It can then be validated by a trusted *proof certificate checker*. The main advantage of this other approach is that it requires a much smaller human effort.

A disadvantage is that every safety claim made by the model checker incurs the cost of generating and then checking the corresponding certificate. This is feasible in general only if such certificates are small and/or simple enough to be checkable by a target proof certificate checker in a reasonable amount of time (say, with at most an order of magnitude slowdown).

This approach allows one to reduce the trusted core to only the proof certificate checker. This allows for a simpler tool verification strategy based on the results of checking the emitted certificates. Section 10 addresses the use of such an approach within the qualification guidance provided by DO-178C and DO-330.

9.1.3 Previous Work on CVC4

The SMT solver CVC4 is a solver for first order propositional logic modulo a set of background theories (like equality, integer arithmetic, arrays, *etc.*). It is one of the most successful such tools and has applications in software and hardware verification, constraint solving, program synthesis, model checking, *etc.* Recent advances have made CVC4 one of the only *proof certificate producing* solvers. A particularity of the certificates that CVC4 generates, compared to other solvers, is that they are *fine grained*. This means they are very detailed and checking them is only a matter of carefully following and replaying the steps in the certificate. This is in contrast to most approaches in the SMT community where the final checker has to perform substantial reasoning to reconstruct missing steps.

The work that we've done in Kind 2 is inspired and relies heavily on the proof production capabilities of CVC4. Proofs in Kind 2 are composed of two main parts:

- Certificates for safety properties of transition systems (which we refer to as the PC)
- Proof certificates that provide evidence that two independent tools accepted the same Lustre input model and produced the same internal representation (which we refer to as the FEC).

9.1.4 Production of the PC

The first certificate summarizes the work of its different engines: bounded model checking (BMC), k-induction, IC3, as well as additional invariant generation strategies. In practice in takes the form of a *k-inductive strengthening of the properties*.

This intermediate certificate is checked by an SMT solver, CVC4, from which we extract proofs to reconstruct safety arguments using the rules of k-induction. Proofs are produced in the language of LFSC.

To make the whole process efficient and scale as much as possible, certificates are first minimized before being checked. An iterative process takes care of this phase by efficiently lowering the bound *k* and removing any superfluous information contained within the certificate.

9.1.5 Production of the FEC

A translation certificate is generated in the form of observational equivalence between two internal representations generated by independently developed front ends. Their equivalence is recast as an invariant property; checking that yields itself a second proof certificate from which a global notion of safety can be derived and incorporated in the LFSC proof. We improve on similar previous approaches by adopting a weaker, property-based notion of observational equivalence, which is enough for our purposes. In practice, we show that the front end of both Kind 2 and JKind agree on their translation (up to weak observational equivalence).

This certificate is turned into LFSC proofs (of equivalence) and can be combined with the previous proofs to form a global safety argument.

9.2 Scope and Purpose

Checking emitted proofs certificates of a model checker is a method for increasing confidence in formal verification tools. The approach has limitations and they are described in this section. Despite these limitations, we still believe the approach provides greater assurance in the correctness of formal methods tools than traditional methods.

The trusted core of our approach consists of:

1. The LFSC checker (5300 lines of C++ code).
2. The LFSC signatures comprising the overall proof system in LFSC (CVC4's sat.plf, smt.plf, th_base.plf, th_int.plf, th_real.plf and our own kind.plf, for k-induction and safety), for a total of 444 lines of LFSC code.
3. The assumption that Kind 2 and JKind do not have identical defects that could escape the observational equivalence check.

We consider point 3 quite reasonable considering the differences between the two model checkers.

A current but temporary limitation of our certificate generation process is that LFSC proofs may contain an unsound proof rule, *trust_f*, which derives any formula. This rule is used by the current version of CVC4 to fill in present gaps in its proof generation code. However, it will be progressively phased out as the instrumentation of CVC4 to produce full proofs is completed.

To overcome this temporary breach, Kind 2 generates sub-goals for these *holes* in a separate LFSC proof file. The proof for these additional goals can be filled manually or by another tool.

9.3 Key Findings

We have confirmed experimentally the usefulness of minimizing certificates. The superiority of full simplification is confirmed by an analysis of the full results. It reduces the size of the invariants on average by 74% (removing on average 19 invariants per certificate) for 42% of the benchmarks. For one benchmark, it removes 236 invariants out of 243. The value of k is reduced in 11% of the benchmarks, by 10 on average, the maximum being a reduction from 36 down to 2.

Finally, we are able to generate and check the proof of invariance for around 80% of benchmarks that Kind 2 succeeds in verifying; we produce and check a complete proof including the front end for 60% of them. Most of the cases where we fail to generate the proof are due to CVC4’s current limitations in its proof producing capabilities. The biggest bottlenecks are the model checking of the equivalence

observer and the simplification of certificates. Despite that, the time cost of the full certification chain is overall within one order of magnitude of the cost of just proving the input property. We find the overall level of performance, which we think we could improve further, already rather good, especially considering that a lot of the benchmarks we used are non-trivial.

9.4 Conclusions

We have devised and implemented a dual technique for generating and checking proof certificates in the model checker Kind 2. Given a Lustre model and one or more invariance properties for it, Kind 2 generates LFSC proofs for the properties it can verify. These proofs have two parts. The first attests that the model and the properties are encoded correctly in Kind 2's internal representation format. It does that by proving the observational equivalence, with respect to the properties, between the internal system and another one produced from the same Lustre input by an independent, third-party tool. The second part attests that the encoded properties are invariants of the internal transition system encoding the Lustre model. Initial certificates are generated as (possibly combined) k-inductive invariants and simplified before being verified by the CVC4 SMT solver. The eventual proof certificates, in LFSC format, are assembled from the proofs generated by CVC4 after verifying these safety certificates.

This approach increases the level of trust that one can place in a model checker like Kind 2. More importantly, because the trusted core of this procedure is mostly reduced to the sole LFSC checker, qualifying this checker (and its proof rules) is sufficient to qualify Kind 2 as a whole. In addition, this qualification is independent of Kind 2 itself as it only depends on the kinds of proofs that are produced. This means that, as long as Kind 2 produces the same kinds of proofs, qualification could be achieved for free regardless of future modifications or improvements in the tool. The qualification of the LFSC checker could even be reused to qualify other model checkers and tools at a greatly reduced cost as long they produce proofs in the same format.

The full technical report is available as a companion document, entitled Proof-Emitting Kind 2 Technical Report.

10 Qualification of Check-It

This activity focuses on the development of a qualification package for a tool that checks the proof certificates emitted by Kind 2, described previously in Section 9, in lieu of qualifying the model checker itself. This approach utilizes a two tool configuration. The first tool, Kind 2, performs model checking on artifacts that represent the design artifact in the DO-178 process. Instead of pursuing the traditional qualification, as described in Section 8, the tool's emitted proof certificates are then passed to a second tool which checks them. This second tool, named Check-It, is an instantiation of the Logical Framework with Side Conditions (LFSC) proof checking framework. Figure 4 illustrates the process.

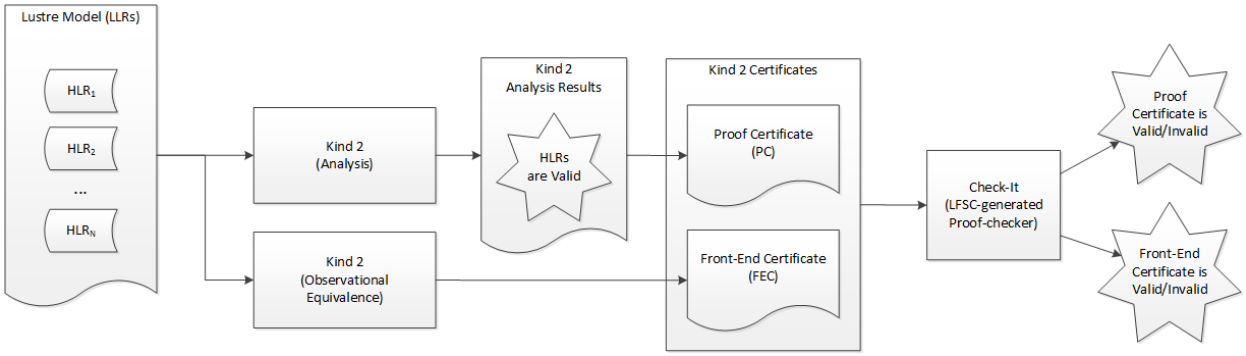

Figure 4 - The two tool configuration

This activity developed a complete tool qualification package for the proof-checking tool Check-It. It examines the two-tool qualification approach and addresses its unique concerns. Similar to the traditional qualification package, this package provides a full set of tool operational requirements, and test cases to demonstrate the tool operates as intended in its operational environment.

10.1 Scope and Purpose

The purpose of this activity is to fully explore the concept of using a qualified proof checking tool to evaluate the output of an unqualified formal methods tool.

This qualification approach considers the suitability of the proof-checking tool to check outputs produced by Kind 2. Check-It is actually an instantiation of the LFSC proof checking framework. LFSC provides a base proof language and infrastructure, but the rules that define the logic for certificates emitted by CVC4 and Kind2 are provided within signature files. The combination of the signature files and LFSC, as shown in Figure 5, define the architecture of Check-It.

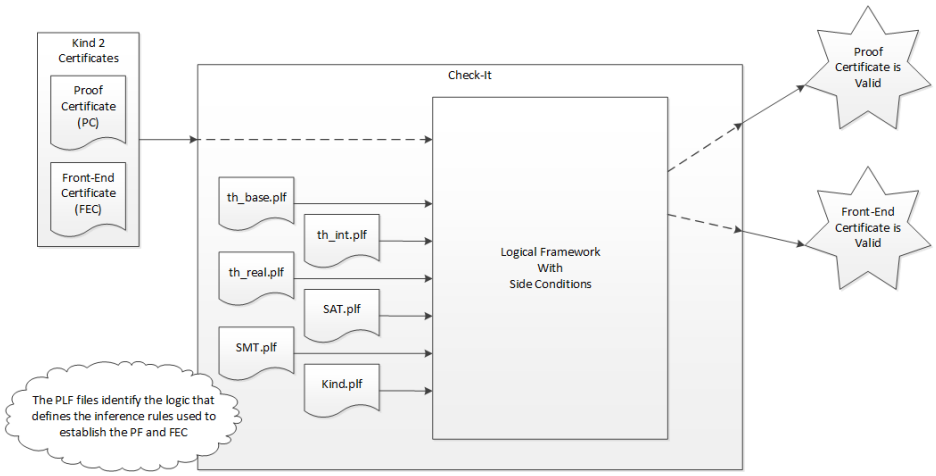

Figure 5 - Architecture of the Check-It Tool

The proof signature files define the logical rules that the Kind 2 is expected to emit. These rules include all of the potential rules that the underlying SMT solver, CVC4, is expected to emit. As such, it is possible to use Check-It to check the results of a proof-emitting SMT solver, but that consideration is out of scope for this effort.

Additionally, Kind 2 emits two certificates. The first certificate, the Proof Certificate, or PC for short, captures the logical rules used to establish the proof of invariance. The second certificate, the Front-End Certificate, or FEC for short, captures a proof that suggests the internal translations performed within Kind 2 are accurate. These two certificates are unique to Kind 2 and Check-It. Check-It is designed to identify and check these two specific certificates for this effort.

10.2 Key Findings

Instrumenting Kind 2 to emit proof-certificates makes it a *verifying* model checker. This means that each time a transition system represented in Lustre satisfies a set of properties, the tool can emit an artifact that is theoretically human reviewable or checkable by a tool. Given the relative size of the typical proof certificate, it is unlikely that humans can adequately review a proof certificate. However, building a proof certificate checker is straightforward; checking proof certificates is a process similar to that of type-checking source code. The goal of this effort is to establish that qualifying a proof checking tool that examines the outputs of a model checker fits within the guidance provided by DO-178C and DO-330.

The benefits of adopting such an approach are numerous. First, a two tool approach provides a greater level of assurance than model checking alone. If a model checker has a software defect and produces an incorrect result, it is not guaranteed that the erroneous results will be discovered quickly, if at all. In a certification environment, this could lead to erroneous results masking the presence of software defects in airborne software. However, by checking the tool's analysis results with a proof certificate checker the possibility of software defects escaping detection is reduced considerably. If the model checker produced an erroneous result then the certificate checker itself would have to also agree with that result. Given that both approaches have dissimilar technical foundations, this is unlikely.

Another benefit of the approach is that a proof checking tool is simpler than a model checker. Check-It is approximately 6000 lines of code (5300 lines, with another 450 lines of code in the signature files). This

is intuitive; it simply parses the provided proof certificate and checks that the certificate is correct. Kind 2 is approximately 50,000 lines of code. It is a parallelized model checker that performs inductive reasoning. It interacts with several instances of an SMT solver in real-time to establish the validity or invalidity of the property. Kind 2 also translates the input file, in the Lustre language, into a First Order Logic (FOL) representation, and performs various optimizations to ensure the analysis returns a result as quickly as possible. The qualification of the simpler tool should result in a smaller, more manageable qualification package as well.

Lastly, a proof checking tool can be built to reason about proofs emitted from multiple tools. In this effort, only the proof certificates emitted by Kind 2 are checked. However, an organization could target the proof language supported by a proof certificate checker with multiple analysis tools, and then elect to only qualify the checker. In theory, one could use any model checker that emits proof certificates and verify its results with one qualified checker. However, the qualification package for the proof certificate checker would have to identify the model checker used and thoroughly examine the interactions between the tools, per DO-330 guidance.

While the proof-checking approach provides additional value over a traditional, testing-based verification approach, it does not fit well within the qualification guidance provided by DO-330. Essentially the model checking tool is used to verify outputs of the DO-178C process and the proof checking tool is used to verify the outputs of the model checker. In fact, the only way to obtain benefit from a two-tool approach is to avoid qualifying Kind 2. Fortunately, the use of a qualified tool to check the results of a qualified tool is addressed in the guidance.

DO-330 FAQ D.7, addresses the use of two tools, one qualified and one unqualified, to obtain certification credit. In fact, the authors of DO-330 anticipated this arrangement, specifically for the use of tools to check the output of a code generator, in lieu of qualifying the code generator itself. FAQ D.7 provides some ground rules for pursuing this approach to ensure the integrity of the two tools results are not impacted. If a qualified tool is used to check the outputs of an unqualified tool the following questions must be considered:

- What are the inputs and outputs of each tool? How can one be sure the unqualified tool is outputting a trustworthy artifact for the qualified tool to examine?
- How do the qualified tool's results relate to the verification objectives satisfied by the qualified tool?
- How do the operating conditions of the qualified tool compare to the potential outputs of the unqualified tool?
- Do the two tools avoid common causes of errors?
- Are there protections in place to avoid adverse interactions between the tools?

The first question is the most difficult to answer. When Kind 2 produces a proof certificate how does the user know the certificate relates to the original Lustre model being analyzed by Kind 2? Check-It does not accept the original Lustre model as input, thus it must trust Kind 2 emits the correct Proof Certificate (PC). The Front-End Certificate (FEC) provides support in that regard; it captures an argument that two tools independently arrived at the same FOL representation of the Lustre model. The FEC is produced by providing the same model to both Kind 2 and JKind [28], a Lustre model checker built in Java. If both tools agree on the FOL representation, then the FEC is built. Thus, in this qualification, if both the PC and

FEC are present, and found to be correct, it is claimed that the unqualified tool's result is trustworthy. This finding is backed by certification experts consulted with during the effort.

The remaining questions are addressed in detail within the Check-It qualification package document, in Sections 3.5.2 – 3.5.5. To summarize, there is nothing about this approach that causes concern. Successfully checking the PC validates the model checker's analysis and checking the FEC provides an argument that the emitted PC corresponds to the original Lustre file. If Kind 2 produces incorrect, malformed, or missing certificates Check-It will highlight the error. The tools use dissimilar technical approaches, one performs model checking and the other proof checking, which is a suitable method for justifying avoidance of common causes of errors. Finally, the tools interact in series which prevents one tool from adversely interacting with the other.

10.3 Conclusions

Qualification of tools, especially those required to be qualified at TQL-5, is a relatively straightforward process and is demonstrated in Section 8 for Kind 2. It requires the user to identify how the tool is used in the software development processes, identify the DO-178C objectives the tool is used to satisfy, and identify the relevant functionality and features of the tool being relied upon.

The process for using Check-It to verify and validate the results from Kind 2 is more complicated, due to more complicated tool architecture, as well as the concerns introduced by DO-330 FAQ D.7. Technically, a two tool approach presents a stronger case for trust than using Kind 2 alone, but the additional concerns about the preservation of trust across two tools also complicates the qualification. Pursuing this approach could be useful in scenarios where multiple tools target the existing language of a proof certificate checker. This would allow one to use multiple tools interchangeably and use Check-It, or some variant, to verify the results of whichever tool was used. For this to work in a production environment a standard proof certificate format should be developed and agreed upon within the community, such that all tools can target a common format.

It is important to stress here that qualifying a model checker isn't a difficult process. Adopting a two-tool approach will require additional work to demonstrate that the unqualified tool cannot insert errors that are undetectable by the qualified tool. Whether or not one decides to pursue a two-tool qualification approach will likely be determined by schedule and budget factors, not technical ones. From a technical perspective, this approach provides stronger evidence for the correctness of the model checker.

The full report is available as a companion document, entitled Check-It Qualification Package.

11 Conclusions

This project thoroughly explores the trust issues associated with formal methods tools within the context of avionics certification. First, the Tool Assurance Study activity is focused on answering the general question of whether a formal methods tool can be trusted. It also provides a systematic mechanism, the trust proposition, for evaluating whether or not a tool and the results it produces are trustworthy. Finally, it highlights specific issues in trusting model checking, theorem proving, and abstract interpretation tools. The produced document provides a thorough and detailed analysis that helps prospective formal methods tool users successfully use formal methods tools.

Remaining project efforts focused specifically on the interpretation of guidance provided by DO-178C and DO-330 for formal methods tools. The Mitigation Strategies task performed a thorough review of the qualification objectives anticipated for formal methods tools. The concepts presented are reinforced in the Tool Qualification Plan case studies that were also developed in the effort. These case studies explore the tool qualification activities for three representative formal methods tools; the model checker Kind 2, the theorem prover PVS, and the Frama-C Value Analysis plug-in based on abstract interpretation. These case studies are designed to identify and satisfy the objectives expected for a TQL-5 qualification. Further, they provide discussion on points of particular interest for each tool. The Kind 2 qualification case study is a full qualification package. It provides a full complement of Tool Operational Requirements and test cases to satisfy them. This package could be used, with some tailoring for the specific application, on avionics software development and certification efforts.

The project also explored the use of alternative verification techniques for tools in a certification context. Traditional verification techniques for tools focuses on testing, review, and analysis to establish that a tool behaves correctly. Efforts to use formal methods to verify tools has been successful, but the downside is they often require significant investment and expertise to perform correctly. Yet another approach is to build a tool such that it emits an artifact that captures a proof of the validity of its results. Such tools are referred to as verifying, or self-verifying. On the project, Kind 2 was extended to emit a proof certificate that capture the logical deduction steps used to establish the proof of invariance of the properties contained within the Lustre model. This certificate can be checked by a second, independent proof certificate checker tool based on the Logical Framework with Side Conditions tool also referred to as LFSC.

Finally, the project developed a qualification package for a tool that checks the proofs emitted by Kind 2. The concept focuses on qualifying a proof certificate checker, which is a smaller, simpler tool, in lieu of qualifying the model checker. This proof certificate checker, which is an instantiation of the LFSC tool, is named Check-It. Check-It will accept proof certificates emitted by Kind 2 and declare them to be valid, or invalid. Much like a human might review a tool's output, Check-It reviews the output of Kind 2. This approach provides a stronger basis of trust in the model checker's results; not only does the model checker reason about the provided properties, but Check-It effectively reviews its work. To escape into production, a software defect would have to go unnoticed by Kind 2 and Check-It. This provides a stronger basis for trust than using Kind 2 alone.

11.1 Key Project Outcomes

This effort not only produced useful artifacts for the formal methods community, but it also provided insight into the qualification process for formal methods tools which will be useful to developers, users,

tool developers, and certification experts. Combined with the prior work in *Formal Methods Case Studies for DO-333*, it provides a comprehensive set of case studies for using and qualifying formal method tools.

This project reveals that qualification at TQL-5 is a straightforward task. The guidance does not require any activities that are difficult or impossible to successfully qualify a model checker. However, the guidance does suggest that tools from the research community may be difficult to qualify at TQL-4 due to the lack of required DO-330 artifacts, including tool requirements, test cases, tool design and architectural descriptions, among others. Formal methods tool developers who are aspiring to have their tools used in the avionics industry should be mindful of this.

In addition, this work highlighted the need for good software engineering practices for formal methods tools used in certification. The relatively high complexity of internal translations, optimizations, and analysis algorithms increases the likelihood that defects will be identified. Bug tracking facilities will be absolutely essential for users to understand what a tool's weaknesses are. Incorrect actions by the user can also impact the trustworthiness of analysis results. Given this, it is important for tool developers to provide adequate training and documentation for potential users.

Lastly, the project developed a proof-emitting model checker, Kind 2, and explored the impact of this technique on tool qualification. It identified a method for using a proof checking tool, named Check-It, to check the results emitted by Kind 2. A qualification package was developed for this approach and reviewed by certification experts.

11.2 Future Work

Guidance on the use of formal methods tools in avionics software certifications was released in 2011 and accepted by the FAA in 2013. While this guidance provided a foundation of principles for using formal methods tools in software certifications, it did not provide guidance on qualification specific to formal methods tools. This project applies the guidance in DO-330 to formal methods tools and addresses this gap. However, there are other issues that remain unaddressed.

First, it is not clear how a tool's analysis scalability will impact the ability to use formal methods for certification. More to the point, if a user can only prove 70% of their formal specifications using formal methods and the remaining portion must be verified using traditional means, is there still benefit? At what point does the cost of carrying two certification strategies outweigh the benefits? Future efforts could focus on answering this question.

Second, the overall cost impact of pursuing a formal methods certification is unknown. If a project decides to pursue a formal methods based certification, what artifacts need to be produced, and what is the anticipated cost of producing them. For example, if Kind 2 is used to verify that a design satisfies its high-level requirements, what is the cost of producing the formalized version of the high-level requirements? There is evidence that the use of formal methods can decrease overall development and verification costs through the early detection and elimination of defects [29]. However, a more comprehensive study in an actual aircraft certification context would be useful in addressing the perceived risks.

The information produced in this effort, and the *Formal Methods Case Studies* effort, produced concrete case studies that will serve the community well. However, the guidance is relatively new, and more work

needs to be done. For example, during the course of this effort, three certification experts were exposed to the qualification packages developed. For two of them, it had been their first exposure to formal methods in a certification setting. While these case studies are useful, a more general awareness campaign, in the form of workshops, etc., could be useful for preparing certification experts to evaluate formal methods based certifications.

Finally, we identify one open question on which to focus future research efforts. Currently, the guidance on using and qualifying formal methods tools is dependent on establishing proofs of correctness. However, there are other techniques that do not provide a proof of correctness but still go above and beyond the capabilities of traditional verification methods. One such example is the use of bounded model checking tools. These tools exhaustively search a given model for a violation of a specification up to a user specified depth. Basically, one can check that a given property is true for the first N steps of a model's execution. More work needs to be done to understand how this can be utilized in a certification setting. The technique is more rigorous than traditional testing for short test vectors, but may not scale up well to produce longer ones. This is an important issue to address because bounded model checking may still be utilized when it is not possible to prove properties of interest. Ultimately it may be viewed as a complementary technique and useful to the overall effectiveness of model checking tools.

12 References

[1] SAE, "Guidelines for Development of Civil Aircraft and Systems," December 2010. [Online]. Available: www.sae.org/technical/standards/arp4754a. [Accessed October 2016].

[2] SAE, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," December 1996. [Online]. Available: www.sae.org/technical/standards/ARP4761. [Accessed October 2016].

[3] RTCA, "DO-254 Design Assurance Guidance for Airborne Electronic Hardware," 19 April 2000. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=752. [Accessed 3 October 2016].

[4] RTCA, "DO-178C Software Considerations in Airborne Systems and Equipment Certification," 15 June 2015. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=803.

[5] RTCA, "DO-330 Software Tool Qualification Considerations," 15 June 2015. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=792.

[6] RTCA, "DO-333 Formal Methods Supplement to DO-178C and DO-278A," 15 June 2015. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=859.

[7] D. Cofer and S. Miller, "Formal Methods Case Studies for DO-333," 2014.

[8] Kind 2 Development team, "Kind 2 Homepage," [Online]. Available: http://kind2-mc.github.io/kind2/. [Accessed 17 September 2015].

[9] G. Hagen and C. Tinelli, "Scaling up the formal verification of Lustre programs with SMT-based techniques," in *Proceedings of the 8th Internal Conference on Formal Methods in Computer-Aided Design*, Portland, 2008.

[10] S. Owre, J. Rushby and N. Shankar, "A Prototype Verification System," in *11th International Conference On Automated Deduction (CADE)*, Saratoga, 1992.

[11] Frama-C, "Frama-C product website," [Online]. Available: http://www.frama-c.com. [Accessed 23 January 2015].

[12] Frama-C, "Frama-C Value Analysis Plug-in," [Online]. Available: http://frama-c.com/value.html. [Accessed 09 July 2015].

[13] RTCA, "DO-278A: Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems," 13 December 2011. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=676. [Accessed 4 October 2016].

[14] RTCA, "DO-331: Model-Based Development and Verification Supplement to DO-178C and DO-278A," 2011 December 2011. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=826. [Accessed 4 October 2016].

[15] RTCA, "DO-332: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A," 13 December 2011. [Online]. Available: http://www.rtca.org/store_product.asp?prodid=848. [Accessed 4 October 2016].

[16] X. Leroy, "Formal verification of a realistic compiler.," *Communications of the ACM,* vol. 52, no. 7, pp. 107-115, 2009.

[17] INRIA, "The Coq Proof Assistant," [Online]. Available: https://coq.inria.fr/. [Accessed 2016 October 2016].

[18] Data61/CSIRO, "sel4 product webpage," [Online]. Available: https://sel4.systems/. [Accessed 2016 October 2016].

[19] University of Cambridge and Technische Universität München, "Isabelle Theorem Prover home page," [Online]. Available: https://isabelle.in.tum.de/. [Accessed 2016 October 2016].

[20] Mathworks, "Polyspace BugFinder™ Product site," Mathworks, [Online]. Available: https://www.mathworks.com/products/polyspace-bug-finder/. [Accessed 16 October 2016].

[21] Mathworks, "Simulink Design Verifier™ Product website," Mathworks, [Online]. Available: https://www.mathworks.com/products/sldesignverifier/. [Accessed 16 October 2016].

[22] A. Biere, A. Cimatti, E. Clarke, O. Strichman and Y. Zhu, "Bounded Model Checking," in *Advances in Computers*, 2003.

[23] M. Sheeran, S. Signh and G. S. Stålmarck, "Checking Safety Properties Using Induction and a SAT-solver," in *Formal Methods in Computer Aided Design*, Austin, 2000.

[24] A. Bradley, "SAT-based model checking without unrolling," in *Proceedings of 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2011.

[25] A. Bradley, "Understanding IC3," in *Theory and Applications of Satisfiability Theory - SAT 2012*, Trento, 2012.

[26] "CBMC Homepage," [Online]. Available: http://www.cprover.org/cbmc/. [Accessed 19 December 2011].

[27] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE,* vol. 79, pp. 1305-1320, 1991.

[28] A. Gacek, "JKind: an infinite state model checker," Rockwell Collins, [Online]. Available: http://loonwerks.com/tools/jkind.html. [Accessed 7 October 2016].

[29] M. Whalen, D. Cofer, S. Miller, B. Krogh and W. Storm, "Integration of formal analysis into a model-based software development process," in *Proceedings of the 12th International conference on Formal Methods for Industrial Critical Systems*, 2007.

[30] T. Kahsai and C. Tinelli, "PKind: A parallel k-induction based model checker," in *Parallel and Distributed Methods in Verification*, 2011.

[31] L. de Moura, H. Rueß and M. Sorea, "Bounded Model Checking and Induction: From Refutation to Verification," in *Computer Aided Verification (CAV) 2003*, Springer Berlin Heidelberg, 2003, pp. 14-26.

[32] L. de Moura and N. Bjorner, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Budapest, 2008.

[33] Wikimedia Foundation Inc, "Satisfiability Modulo Theories Wikipedia Page," Wikimedia Foundation Inc, 14 June 2015. [Online]. Available: https://en.wikipedia.org/wiki/Satisfiability_modulo_theories. [Accessed 2 July 2015].

[34] E. Clarke, A. Biere, R. Raimi and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," *Formal Methods in System Design,* vol. 19, pp. 7-34, 2001.

[35] CVC4 Team, "CVC4: the smt solver," [Online]. Available: http://cvc4.cs.nyu.edu/web/. [Accessed 6 August 2015].

[36] SRI, "Yices SMT Solver," [Online]. Available: http://yices.csl.sri.com/. [Accessed 6 August 2015].

[37] Microsoft Research, "Z3 SMT Solver," [Online]. Available: https://github.com/Z3Prover/z3. [Accessed 6 August 2015].

[38] A. Bradley, "IC3 and beyond: Incremental, Inductive Verification," in *Computer Aided Verification*, Berkeley, 2012.

[39] A. Cimatti, A. Griggio, S. Mover and S. Tonetta, "IC3 Modulo Thoeries via Implicit Predicate Abstraction," in *Tools and Algorithms for the Construction and Analysis of Systems - TACAS 2014*, Grenoble, 2014.

[40] A. Cimatti and A. Griggio, "Software Model Checking via IC3," in *Computer Aided Verification - CAV 2012*, Berkeley, 2012.

[41] A. Stump, D. Oe, A. Reynolds, L. Hadarean and C. Tinelli, "SMT proof checking using a logical framework," *Formal Methods in System Design,* vol. 42, no. 1, pp. 91-118, 04 July 2012.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 02- 2017	Contractor Report	

4. TITLE AND SUBTITLE

Formal Methods Tool Qualification

5a. CONTRACT NUMBER

NNL14AA06C

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Wagner, Lucas G.; Cofer, Darren; Slind, Konrad; Tinelli, Cesare; Mebsout, Alain

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

999182.02.85.07.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA-CR-2017-219371

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 64
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Benedetto L. Di Vito

14. ABSTRACT

Formal methods tools have been shown to be effective at finding defects in safety-critical digital systems including avionics systems. The publication of DO-178C and the accompanying formal methods supplement DO-333 allows applicants to obtain certification credit for the use of formal methods without providing justification for them as an alternative method. This project conducted an extensive study of existing formal methods tools, identifying obstacles to their qualification and proposing mitigations for those obstacles. Further, it interprets the qualification guidance for existing formal methods tools and provides case study examples for open source tools. This project also investigates the feasibility of verifying formal methods tools by generating proof certificates which capture proof of the formal methods tool's claim, which can be checked by an independent, proof certificate checking tool. Finally, the project investigates the feasibility of qualifying this proof certificate checker, in the DO-330 framework, in lieu of qualifying the model checker itself.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	45	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658